Project #2

CSC 261/461 (Database Systems), Fall 2018,
University of Rochester
Due Date: 11/10/2018 (11:59 pm)
This is NOT a group project

This project contains two parts. It acts as an example of transforming raw data into a MySQL database. Please start early – it is not trivial.

PART 1

Introduction

We provide you with a fairly large volume of data downloaded (over a decade ago!) from the eBay website stored as XML files. XML format is primarily used to store semi-structured data. We also provide you a parser, `parser.py`, which converts these xml files into four structured `.dat` files for loading into MySQL database.

Your task is to examine the `.xml` and `.dat` files, and provide an ER model and a schema for the database. Then, load the transformed data (.dat files) into MySQL database and test it by running some SQL queries over it.

Task A: Examine and transform the XML data files

We are providing an XML auction data set for you to use in your project. The data files are located in the directory `~csc261/proj2/data/`. You can access this folder from `betaweb.csug.rochester.edu`. This AuctionBase database is consist of 40 files: `items-0.xml, items-1.xml, ..., items-39.xml`. You do not need to copy these files. But you should open a few of these files and look into the pattern or structure to familiarize yourself with the schema of the dataset.

There are a total of about 20,000 auctions. Note that the **ItemID** attribute is unique and involved in only one auction, while the **Name** attribute is not unique. One of the important aspects to understand about the data is that it represents a single point in time. (Specifically, it represents the following point in time: December 20th, 2001, 00:00:01.) It contains items that have been auctioned off in the past and items that are currently up for auction.

Here is a sample of the data:

```
<Item ItemID="1043402767">
<Name>Precious Moments Girl Stove Mini Tea Set</Name>
<Category>Collectibles</Category>
<Category>Decorative &amp; Holiday</Category>
<Category>Decorative by Brand</Category>
<Category>Enesco</Category>
<Category>Precious Moments</Category>
<Currently>$4.00</Currently>
<First_Bid>$4.00</First_Bid>
<Number_of_Bids>1</Number_of_Bids>
<Bids>
<Bid>
<Bidder UserID="goldcoastvideo" Rating="2919">
<Location>Los Angeles,CA</Location>
<Country>USA</Country>
```

```
        </Bidder>
        <Time>Dec-06-01 06:44:54</Time>
        <Amount>$4.00</Amount>
        </Bid>
        </Bids>
        <Location>MIlwaukee Wi</Location>
        <Country>USA</Country>
        <Started>Dec-03-01 18:44:54</Started>
        <Ends>Dec-13-01 18:44:54</Ends>
        <Seller UserID="fallsantiques" Rating="952"/>
        <Description>PRECIOUS MOMENTS GIRL/STOVE MINI TEA SET:</Description>
        </Item>
```

You need to convert this unstructured data into a structure for loading into MySQL database. We are giving you the utility `parser.py` for that.

You can execute the following commands after you log in to Betaweb machine to copy `parser.py` and run it to get the structured .dat files.

```
        mkdir proj2
        cd proj2
        cp ~csc261/proj2/parser.py .
        python parser.py ~csc261/proj2/data/*.xml
```

This parsing would take some time and finally produce four `.dat` files. After the parsing, your directory should have four .dat files. `item.dat, user.dat, category.dat, and bid.dat`

Open these files to view their contents. Each of these files contains multiple fields separated by <>.

- `item.dat` contains nine fields (itemID, name, currently, buy_price, first_bid, started, ends, userID, and description)
- `user.dat` stores four fields (userID, rating, location, and country)
- `category.dat` stores only two fields (itemID, and category).
- `bid.dat` contains four fields (itemID, userID, time, and amount).

Task B: Design your relational schema

The four files that `parser.py` provides will eventually be loaded as four relations. Just to provide a basic understanding of this database to non-technical people, you need to provide an ER model of the data.

Your ER model should have a number of entities. Draw their attributes and how these entities are related to each other. Provide the cardinality and participation constraint (total vs partial), and (min, max) constraint. Your ER model should be in Chen Notation.

Provide your relational schema definitions in text form, including the attributes and their data types for each relation. Make sure to indicate your chosen keys (including primary and foreign).

Create a pdf file containing the ER model and the database schema and save it as `proj2p1report.pdf`.

While designing your relational schema, you may realize that two fields from the XML files are technically not necessary to have in your schema for it to represent the same information: Currently and Number of Bids. For example, the Number of Bids can be obtained by simply running a query to calculate the number of bids on a particular item. However, for the purposes of this project, ignore Number of bids and keep only 'currently' in your schema. (`user.dat` also ignores numberOfBids). For websites with large databases and many users, running a query to calculate an aggregation every time it needs to be viewed can be costly, so storing the aggregated result itself can help improve performance.

Task C: Load your data into MySQL

The next step is to create and populate your AuctionBase database. MySQL provides a facility for reading a set of commands from a file. You should use this facility for (re)building your database and running sample queries, and you must use it extensively in your submitted work.
Create a command file called `create.sql` that includes the SQL commands that create all of the necessary tables according to your schema design from Task B. Before creating the tables, you should also ensure that any old tables of the same name have been deleted. (This makes it easier to test your submission.) This file will look something like:

```
drop table if exists User;
drop table if exists Item;
...
create table User ( ... );
create table Item ( .... );
...
```

Please use the **exact names for your tables**: `User`, `Item`, `Bid` and `Category`, since our test program will assume that you are using those table names.
Then, create a command file called `load.sql` that loads your data into your tables. This file will look something like:

```
...
LOAD DATA LOCAL INFILE "user.dat"
INTO TABLE User
FIELDS TERMINATED BY "<>";
...
```

If you are unsure what these commands do, refer to `https://dev.mysql.com/doc/refman/5.7/en/load-data.html`

It is possible to execute SQL statements from a text File. Refer to `https://dev.mysql.com/doc/refman/5.7/en/mysql-batch-commands.html` for details. In a nutshell, If you are already running MySQL, you can execute an SQL script file using the source command. For example,

```
mysql> source create.sql
```

Execute sql statements from `create.sql` and `load.sql` to create and populate these tables.

Task D: Test your MySQL database

The final step is to take your newly loaded database for a test drive by running a few SQL queries over it. As with database creation, first test your queries interactively using the MySQL command-line client, then store these queries in a file to execute them using source command. First, try some simple queries over one relation, then more complex queries involving joins and aggregation. Make sure the results look correct. When you are confident that everything is correct, write SQL queries for the following specific tasks:

1. Count the number of Items from category "Enesco"
2. Count the number of Items (same as auctions) belonging to exactly four categories.
3. Find the ID(s) of auction(s) with the maximum buy_price.
4. Count the number of sellers whose rating is higher than 1000.

5. Count the number of users who are both sellers and bidders.
6. Count the number of categories that include at least one item with a bid of more than $1000.
7. Count the number of pair of bidders, who bid for the same item.
8. Count the number of sellers who sell items in more than 10 categories

Your answers to the above eight queries over the large data set should be (in order):
`349, 8365, 1677348181, 3130, 6717, 4, 24668, and 234.`

Your queries should be fairly efficient – they should each take at most ten seconds to execute. If any of your queries is taking much longer than that, you've probably written them in an unnatural way; please try rewriting them, and see the course staff if you need any help. Also, this is a good time to attend the workshops and meet TAs. Also, you can make appointments with any TA.

Put each of the eight queries in its own command file: query1.sql, query2.sql, ..., query8.sql. Make sure that the naming of your files corresponds to the ordering above, as we will be checking correctness using an automated script on a different data set.

PART 2

Introduction

This part of the project will make use of MySQL triggers to monitor and maintain the integrity of your AuctionBase data. You will also add a "current time" feature to your AuctionBase database. Your project must be implemented in MySQL and validated on the betaweb machines.

Task A: Adding Support for Current Time

The original auction data that we provided for you in XML, which you translated into relations and loaded into the database in Part 1, represents a single point in time, specifically, one second after midnight on December 20th, 2001 (represented in MySQL as 2001-12-20 00:00:01). To fully test your functionality, and to simulate the true operation of an online auction system in which auctions close as time passes, you should maintain a fictitious "current time" in your database. First, add a new one-attribute table to your database that represents this current time. For Consistency, let's name this table and the attribute CurrentTime and timeNow.

This table should at all times contain a single row (i.e., a single value) representing the current time of your AuctionBase system.

For starters, the table should be initialized to match the single point in time we've previously mentioned: 2001-12- 20 00:00:01. To do this, modify your create.sql from Part 1 by adding the necessary SQL commands to create and initialize your `CurrentTime` table. Also, to make sure that you've initialized everything correctly, include ac SELECT statement that reads the current time of your AuctionBase system. Your `create.sql` should now include the following:

```
DROP TABLE if exists CurrentTime ;
CREATE TABLE CurrentTime (timeNow DATETIME NOT NULL) ;
INSERT INTO CurrentTime values ('2001-12-20 00:00:01') ;
SELECT timeNow FROM CurrentTime ;
```

Task B: Adding Constraints and Triggers to Your Schema

For this task, you may want to refer to the MySQL documentation for the CREATE TRIGGER and DROP TRIGGER statements `https://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html`.

To ensure that the AuctionBase system at a given point in time represents a correct state of the real world, a number of real-world constraints are expected to hold. In particular, your database schema must adhere to the following constraints:

Primary Key and Foreign Key constraints

- No two users can share the same User ID.
- All sellers and bidders must already exist as users.
- No two items can share the same Item ID.
- Every bid must correspond to an actual item.
- An item cannot belong to a particular category more than once.
- All sellers and bidders must already exist as users.
- No user can make a bid of the same amount to the same item more than once.

- All this constraints can be imposed using Primary Key and Foreign Key constraints. If you have not done this already in Part 1, please modify `create.sql` file to incorporate these constraints.

Constraints achieved by Triggers (You need to create Triggers)

Constraints for Bidding

1. A user may not bid on an item he or she is also selling.
2. No auction may have two bids at the exact same time.
3. No auction may have a bid before its start time or after its end time.
4. No user can make a bid of the same amount to the same item more than once.
5. Any new bid for a particular item must have a higher amount than any of the previous bids for that item.
6. All new bids must be placed at the time which matches the current time of your AuctionBase system. item.

Constraints for Items

7. The Current Price of an item must always match the Amount of the most recent bid for that item. (Note: This can be updated when a new bid for that item is inserted)

Constraints for Time

8. The current time of your AuctionBase system can only advance forward in time, not backward in time.

- We are only concerned about two trigger events, when a new bid is inserted, and when the current time gets updated. The current version of MySQL does not support two separate triggers for the same event at same trigger time. (We can have two different triggers for the same event if the trigger time is different. For example, Trigger for Constraint 7 should only get activated after a successful entry of a Bid). For this part, you need to create three (3) triggers to implement this 8 constraints.
- Also, this version of MySQL does not support SIGNAL command. A workaround for this is 'call'ing a function that does not exist, and the name of this 'non-existing' function gives us clue about the error!

To make it simple for you, we are providing you a sample trigger for the following constraint:
The end time for a bid must always be after its start time.

```
# Constraint: The end time for a bid must be after its start time.
DROP TRIGGER IF EXISTS item_check_before;

DELIMITER //
CREATE TRIGGER item_check_before BEFORE INSERT ON Item
FOR EACH ROW
BEGIN
IF NEW.ends < NEW.started THEN
```

```
            CALL `Error0:The end time must be after its start time.`;
            END IF;
            END;//
            DELIMITER ;
```

Create a file, `create_triggers.sql` which should contain code for creating triggers to satisfy all these constraints. Please make sure you drop the existing trigger before creating a new one (or creating after modification)

For automated grading, we mandate the following naming scheme for the triggers:

- **before_insert_bid**
- **after_insert_bid**
- **before_update_time**

Also, Number the Error Messages from 'Error1:< description of the error>' to 'Error8:<description of the error>' for Constraints 1-8. Note, You do not need to have any 'Error7: ...' as we are only updating a field in another table.

Task C: Test the triggers

Run some sample queries after creating the triggers to see if the triggers are reporting errors or updating fields as expected.

It is possible to create all the files, load them and then create all the triggers using the source command as we have done in Project 2 Part 1. For example,

```
            mysql> source create.sql
            mysql> source load.sql
            mysql> source create_triggers.sql
```

For avoiding any deduction your submission must need to follow the following guidelines:

- 'timeNow' is the only attribute in your CurrentTime relation which holds the value '2001-12-20 00:00:01' in DATETIME format. The code for creating and loading the relation is in `create.sql` file.
- All the Primary Key and Foreign Key constraints hold.
- All the triggers in `create_triggers.sql` work.
- You are numbering the error messages based on the constraint numbers (from 1 to 8) and your error messages are descriptive.
- You are following the naming convention for the triggers.
- The constraints are numbered based on their priority. For any query, if two constraints i and j are violated, then Error$i$ should be reported if $i < j$.

Once all the requirements are satisfied, follow the submission instruction.

Submission instructions

To submit the project, first gather the following files in a single submission directory `proj2`:

```
    proj2p1report.pdf
    create.sql
    load.sql
    query1.sql
    query2.sql
    query3.sql
    query4.sql
    query5.sql
    query6.sql
    query7.sql
    query8.sql
    create_triggers.sql
```

Now, create an archive file (with .tar extension) for submission.

```
    cd ..
    tar -cvf proj2.tar proj2
    ls
```

(Note: `cd ..` takes you to the parent directory and `ls` command should display all the files int the directory including the tar file.)

Submit the tar file using the command (Note: Don't forget the tilde (~) at the beginning):

```
    ~csc261/submit p2 proj2.tar
```

You should get a feedback whether the submission is successful or not.

You may resubmit as many times as you like; however, only the latest submission (along with the timestamp) will be saved, and we will use your latest submission for grading your work. Submissions via email will not be accepted! No late submissions allowed.

Grading

Total: 120 pts

Report (30 pts)

- (10 pts) Written schema
  You must include all the attributes and their datatypes. Also, include the primary key and the foreign key(s) (if any) for each relation. If you have a foreign key, you must show which field it references to.

- (20 pts) ER Diagram
  You must include all the entity sets, their attributes, and all the relations. Also, you must provide the cardinality, participation constraint (total vs partial), and(min, max) constraint.

  Note that though only providing (min, max) constraint is enough and providing cardinality and participation constraint is redundant, but for mastering the concepts you MUST provide all, cardinality, participation constraint (total vs partial), and(min, max) constraint.

SQL (50 pts)

5 pts for each of the following files:

```
              create.sql
              load.sql
              query1.sql
              query2.sql
              query3.sql
              query4.sql
              query5.sql
              query6.sql
              query7.sql
              query8.sql
```

Triggers (40 pts)

5 pts for each trigger in `create_trigger.sql`.

Academic Honesty

You MUST NOT share (or get) any component of this project with (or from) any other students which includes your Project 1 team members. This project should be solely your own work. Once you master the material, you are encouraged to share your learning while collaborating in Project 1.

For more details, see: `https://www.rochester.edu/college/honesty/`

Acknowledgement

Adapted from a project given in CS145 offered by Stanford University.